APS Course Project

Implementing Suffix Tree in O(n) time

Member 1 :Ashish Kempwad(2019201091)
Member 2 :Pulkit Sharma(2019201006)

1. Introduction

1.1 Motivation:

Suffix Tree is very useful in numerous string processing and computational biology problems.A suffix tree **T** for a m-character string S is a rooted directed tree with exactly m leaves numbered 1 to **m.** (Given that last string character is unique in string). Suffix tree comes very handy in numerous string operation and execution. We here try to tackle it in O(n) time using a popular **Ukkonen's algorithm**.

1.2 High Level Ukkonen's algorithm

Construct tree T_1

For i from 1 to m-1 do
begin {phase i+1}
For j from 1 to i+1
begin {extension j}
Find the end of the path from the root labelled S[j..i] in the current tree.
Extend that path by adding character S[i+l] if it is not there already
end;
end;

2. Extension Rules:

2.1 **The 3 extension rules:**

Rule 1: If the path from the root labelled S[j..i] ends at leaf edge (i.e. S[i] is last character on leaf edge) then character S[i+1] is just added to the end of the label on that leaf edge.

Rule 2: If the path from the root labelled S[j..i] ends at non-leaf edge (i.e. there are more characters after S[i] on path) and next character is not s[i+1], then a new leaf edge with label s{i+1] and number j is created starting from character S[i+1].
A new internal node will also be created if s[1..i] ends inside (in-between) a non-leaf edge.

Rule 3: If the path from the root labelled S[j..i] ends at non-leaf edge (i.e. there are more characters after S[i] on path) and next character is s[i+1] (already in tree), do nothing.

2.2 **Suffix tree Node structure and various details:**

We will have SuffixTreeNode structure to represent each node in tree. SuffixTreeNode structure will have following members:

- **children** – This will be an array of alphabet size. This will store all the children nodes of current node on different edges starting with different characters.
- **suffixLink** – This will point to other node where current node should point via suffix link.

- **start, end** – These two will store the edge label details from parent node to current node. (start, end) interval specifies the edge, by which the node is connected to its parent node. Each edge will connect two nodes, one parent and one child, and (start, end) interval of a given edge will be stored in the child node. Lets say there are two nods A (parent) and B (Child) connected by an edge with indices (5, 8) then this indices (5, 8) will be stored in node B.
- **suffixIndex** – This will be non-negative for leaves and will give index of suffix for the path from root to this leaf. For non-leaf node, it will be -1 .

2.3 **Representation of Suffix Tree using Ukkonen's :**

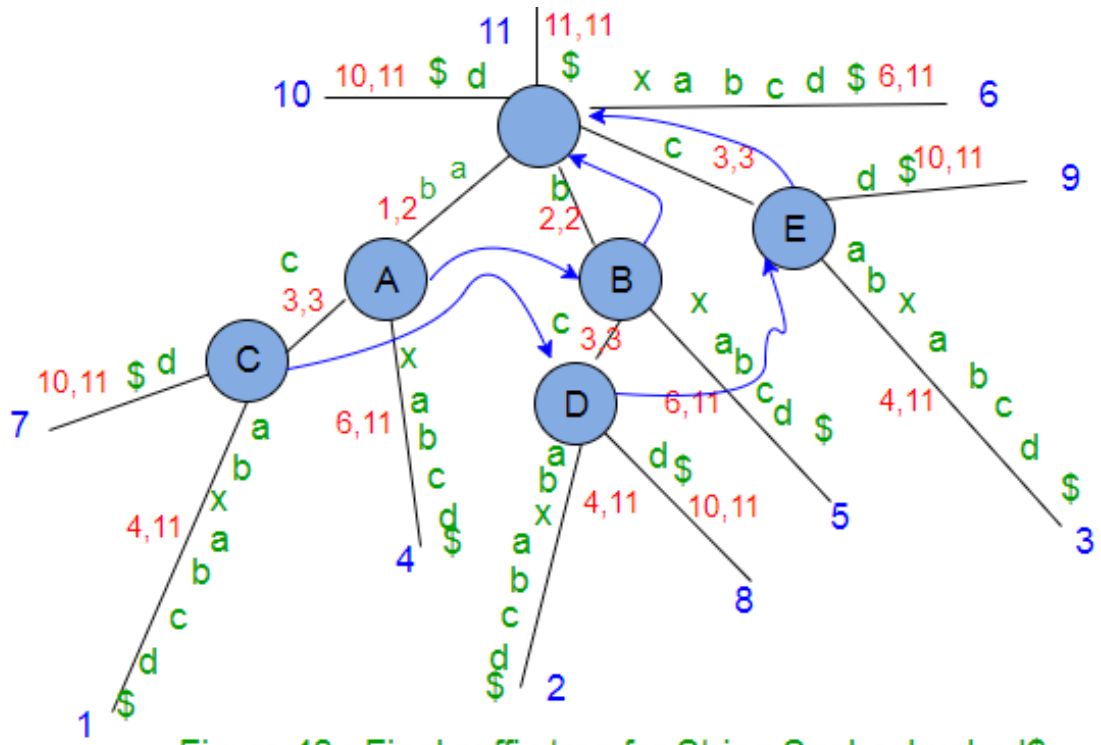
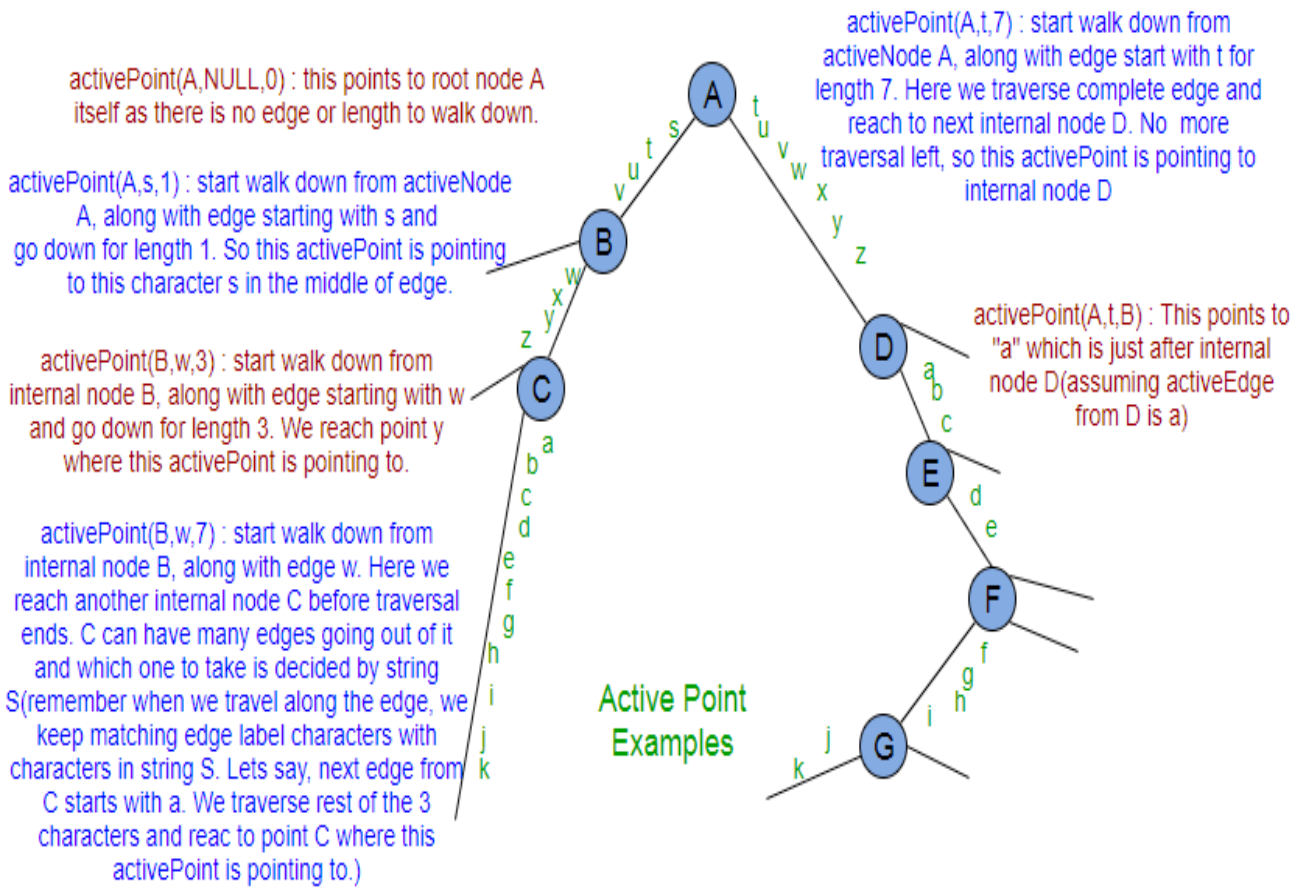


Figure 43 : Final suffix tree for String S=abcabxabcd$

3. Structure of suffix tree and active point example :-

```
struct Suffix
{
    struct Suffix *node[MAX_CHAR];
    struct Suffix *suffixLink;
    int start;
    int *end;

    int suffixIndex;
};
```



3.1 Active points:-

```
int remaining = 0;
// active points
int active_edge = -1;
int active_length = 0;
int string_size = -1;
```

4. Applications:

1. Substring check:

Ukkonen’s Suffix Tree Construction takes O(N) time and space to build suffix tree for a string of length N and after that, traversal for substring check takes O(M) for a pattern of length M.

2.Searching all pattern:

Ukkonen’s Suffix Tree Construction takes O(N) time and space to build suffix tree for a string of length N and after that, traversal for substring check takes O(M) for a pattern of length M and then if there are Z occurrences of the pattern, it will take O(Z) to find indices of all those Z occurrences.
Overall pattern complexity is linear: O(M + Z).

3.Longest Repeated Substring:

Ukkonen’s Suffix Tree Construction takes O(N) time and space to build suffix tree for a string of length N and after that finding deepest node will take O(N).
So it is linear in time and space.